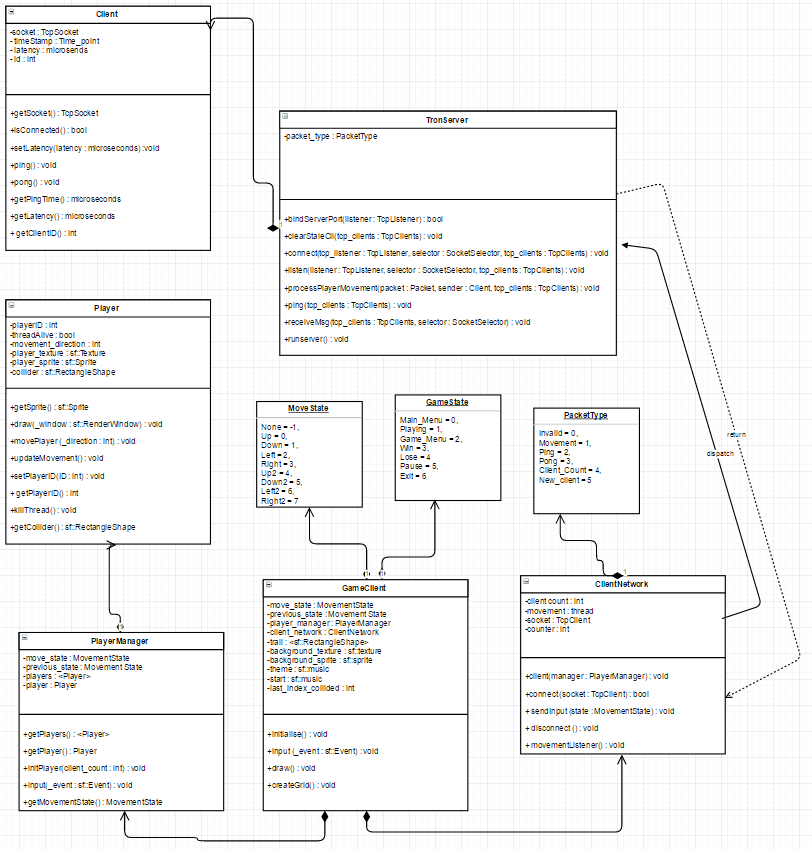
LOW-LEVEL PROGRAMMING DOCUMENTATION

# Class Hierarchy



# Justification

The above details the class structure (UML Diagram) for my Tron project. The main game is handled within the **GameClient** class; containing information about the game window, and is responsible for looping through objects and ticking/rendering them. The class also includes a pointer to both the **ClientNetwork** and **PlayerManager** to handle connecting to the server for the handling of connections and packets and retrieving information about all of the connecting players respectively.

**ClientNetwork** connects directly to the server located at **TronServer** and sends appropriate information while listening for incoming packets and handles them depending on their **Packet\_Type** (obtained from the enum).

**PlayerManager** initializes and sets up individual **players** when they connect to the server. Furthermore, itcontains the vector that all connected players are stored within. As well as handles the input when instructed to, based on packets received in **ClientNetwork.**

**Player** is the games way of treating connected sockets as game objects. It contains all the data exclusively useful to an individual player.

**TronServer** is the hosted server, which contains a pointer to **Client**, which contains data about the connections to the server.

Finally, the code returns to **GameClient** which renders a gamegrid, and handles the logic individual tiles, including colouring them to the individual player and handling collisions.

# Post-Mortem

In hindsight, I found it incredibly easy to capture an enjoyable and interesting theme for the Tron game. I had the idea right from the start which made applying it as I developed the game incredibly easy. Unfortunately, I ran into technical troubles rather frequently along the way that prevented me from developing as fast and as concisely as I’d like.

It took a long time for networking and packets to click, which ultimately resulted in a lot of delayed work on the project, leaving less time to work on at the end. I could have approached the project from a different way and began to work on other elements outside of networking in the meantime, but I felt that the result would have made the task much bigger in the end.

Contrastingly, I set up threading very early on for my draw function initially, as I had an issue where the game wouldn’t progress because it was stuck polling the draw loop. I later extended this to thread input and movement so that the game and the other player were never waiting for movement to complete before they could act.

Thankfully, I managed to have my moment virtually entirely game/server sided which meant there was little room for client “hacking” to modify values and give an unfair advantage. I should have applied delta time to the values I used for movement for much more consistent results, but I struggled with passing the value through my threaded movement function.

My approach to line drawing was relatively successful, in creating a grid of rectangles whose fill colour would be changed when colliding with an object. Sadly it led to an unresolved bug with two players instantly colliding (with their attached ‘collider’ rectangles) due to the spawn location of the attached triangles. I didn’t really invest much time into thinking about alternate approaches and I think that if I had, I may have found a more reliable way to handle the entire system.

However, collision does work and ends the game if only one client is connected, it isn’t an ideal solution but I found this to be a good compromise rather than no collision for one or two players.

I definitely should plan more in future, sticking to a design pattern and having a consistent design philosophy would help remove a lot of complications I face, and its certainly something I’ll be revisiting and considering more for future tasks.